NAWCWD TP 8590
NAVAIR

Bi-Conjugate Gradient Algorithm for Solution of Integral Equations Arising in Electromagnetic Scattering Problems

by
Surendra Singh
University of Tulsa

Klaus Halterman and J. Merle Elson
Research Department

SEPTEMBER 2004

NAVAL AIR WARFARE CENTER WEAPONS DIVISION
CHINA LAKE, CA 93555-6100

Naval Air Warfare Center Weapons Division

FOREWORD

This report details the implementation of applying the bi-conjugant gradient algorithm to solve a volume integral equation. The integral equation is a solution to Maxwell's equations and involves scattering of a plane wave from a metallic nano-cyclinder. The solution algorithm is outlined, a numerical example is given, and a FORTRAN code listing is provided. This work was done during a 2-month period at the Naval Air Warfare Center Weapons Division (NAWCWD), China Lake, California, during June and July 2004. The ONR-ASEE Fellowship Program funded it.

This report was reviewed for technical accuracy by Pamela L. Overfelt of the Sensor and Signal Sciences Division, Research Department, NAWCWD Code 4T4100D.

Approved by
C. F. MARKARIAN, *Head*
Research Department
2 August 2004

Under authority of
D. J. VENLET
RADM, U.S. Navy
Commander

Released for publication by
KAREN HIGGINS
Director for Research and Engineering

NAWCWD Technical Publication 8590

Published by . . . . Technical Information Division
Collation . . . . Cover, 11 leaves
First printing . . . . 30 copies

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, D.C. 20503.			
1. AGENCY USE ONLY *(Leave Blank)*	2. REPORT DATE September 2004	3. REPORT TYPE AND DATES COVERED June-July 2004	
4. TITLE AND SUBTITLE Bi-Conjugate Gradient Algorithm for Solution of Integral Equations Arising in Electromagnetic Scattering Problems (U)			5. FUNDING NUMBERS
6. AUTHOR(S) Surendra Singh, Klaus Halterman, and J. Merle Elson			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Air Warfare Center Weapons Division China Lake, CA 93555-6100			8. PERFORMING ORGANIZATION REPORT NUMBER NAWCWD TP 8590
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Office of Naval Research 1900 N. Quincy Ave. Arlington, VA 22201			10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES			
12a. DISTRIBUTION/AVAILABILITY STATEMENT A Statement; public release; distribution unlimited.			12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

(U) The bi-conjugate gradient (bi-CG) algorithm is applied to numerically solve linear equation systems resulting from integral equations arising in electromagnetic scattering problems. The basic advantage of using this algorithm over traditional methods, such as matrix inversion, is that the algorithm is iterative in nature. The iterative nature allows the user to control the residual error in the final solution. Also, the algorithm can be implemented without storing the coefficient matrix, thus providing huge saving in storage requirements. It was realized that the existing code that utilized matrix inversion to solve the linear equation system was limited to a coarse discretization of the geometry. This code could not handle very fine geometry discretization due to storage limitations. With the implementation of the bi-CG algorithm, this limitation was overcome. The present code can very easily handle very fine discretizations, thereby vastly improving the utility of the computer code. This report outlines the bi-CG algorithm and provides its implementation to solve an electromagnetic scattering problem of a nanowire illuminated by a plane wave. The report also includes a complete FORTRAN listing of the code.

14. SUBJECT TERMS Iterative Methods, Integral Equations, Electromagnetic Scattering, Bi-Conjugate Gradient Algorithm			15. NUMBER OF PAGES 21
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	19. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	20. LIMITATION OF ABSTRACT SAR

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

Standard Form 298 Back (Rev. 2-89)

SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

CONTENTS

INTRODUCTION

The work described in this report was performed in collaboration with Professor Surendra Singh, who came to the Naval Air Warfare Center Weapons Division (NAWCWD), China Lake, California, during June and July 2004. Professor Singh, a faculty member in Electrical Engineering Department at The University of Tulsa, Tulsa, Oklahoma, was visiting as an Office of Naval Research-ASEE Summer Faculty Fellow. He worked in the Optics, RF and Material Physics Branch, Sensor and Signal Sciences Division, Research Department. This work involved learning and implementing the bi-conjugate gradient algorithm in order to numerically solve linear equation systems resulting from integral equations arising in electromagnetic scattering problems.

The bi-conjugate gradient (bi-CG) algorithm (Reference 1) is a well-known and established iterative technique to solve a system of linear equations of the form $\mathbf{A}\,\mathbf{x} = \mathbf{b}$. In this equation the square matrix $\mathbf{A}$ and the source or excitation vector $\mathbf{b}$ are both complex. The algorithm converges most rapidly if the matrix $\mathbf{A}$ is square, symmetric, and positive-definite. This does not imply that the algorithm is not suitable if the matrix is not symmetric. We have applied the algorithm to non-symmetric matrices with good results. Details of the bi-CG algorithm are provided in the next section. A variant of this algorithm, the bi-CG stabilized (bi-CGSTAB) algorithm (Reference 2), was also implemented. But in the numerical experimentation, we found that the stabilized version did not provide any advantage over the earlier version. A very good source for a basic understanding of the conjugate gradient method is provided in Reference 3. In the Bi-Conjugate Gradient Method section, we provide the solution of an integral equation known as the Lippman-Schwinger equation. The 2-dimensional physical problem involves a plane wave incident on a cylinder. Note that the original computer code to obtain the numerical solution of the integral equation was developed by Klaus Halterman and J. Merle Elson at China Lake. This code utilized matrix inversion to solve the system of equations. The code was modified to accommodate the bi-CG algorithm. A listing of the FORTRAN code of the bi-CG algorithm and associated function subprograms is provided in the Appendix.

BI-CONJUGATE GRADIENT METHOD (bi-CG)

The bi-CG method is an iterative technique used to solve a system of linear equations of the form **A x = b**. The method is most suitable in situations where matrix **A** is so large that storing the matrix may pose a significant storage problem. This means that the method can be implemented without storing the entire **A** matrix. This is accomplished by just calculating a single row or column of the matrix at a time. This provides tremendous savings in storage requirements and allows the user to solve a very large system of equations. Note that the savings in storage requirements comes at the cost of increased computation time because each iteration of the algorithm requires computing the elements of matrix **A** twice. In order to increase the computational efficiency in evaluating the matrix elements, we found that the matrix elements involve the computation of two types of Hankel functions. The two Hankel function values are computed once and stored in two arrays. The arrays are then utilized in computing the matrix elements. This technique provided considerable saving in computation time. It is observed that the bi-CG algorithm converges a little slower when the matrix **A** is dense, and that the convergence is faster when the matrix **A** is sparse. The pseudo-code for the bi-CG method is given below:

Initialize the following variables: conv $=10^{-4}$, rerr = 100, $\beta = 0$, and the following vectors: r = b, rn = 0, x = 0, p = 0, pn = 0, t = 0.

```
rn = r*  (* denotes complex conjugate)
val1 = b*b

do i = 1,nitm
    if rerr>conv
        val2 = r*r
        rerr = abs(val2/val1)
        val3 = rn*r
        if (i≠1)  $\beta$ = val3/val5
        p = r + $\beta$ p
        pn = rn + $\beta$ pn
        t = A p          (This operation can be performed by computing one
row of A at a time, thus avoiding the storage of matrix A.)

        val4 = pn*x
        $\alpha$ = val3/val4
        x = x + $\alpha$ p
```

```
                $r = r - \alpha t$
                t = transpose(**A***) pn        (This operation can be performed by
computing one column of the matrix **A** at a time, thus avoiding the storage of the matrix
**A**.)
                $rn = rn - \alpha t$
                val5 = val3
            end if
    end do
```

SUBROUTINE bcg (b,nunks,nitm,conv,ci,nit,rerr)

This subroutine iteratively solves a system of linear equations of the form **[A][ci]=[b]** using the bi-CG method. The subroutine does not require the storage of the matrix **A**. Rather it uses two function programs, *arow* and *acol* which provide a specific row or column of the matrix, respectively. Note that vector **x** is replaced by **ci**.

Input Variables:

nunkns (integer) - Total number of unknowns

b (complex vector) - Righthand side or excitation vector of length *nunkns*

nitm (integer) - Maximum number of iterations for bi-CG (typically set it equal to *nunkns*)

conv (real) - Convergence factor for bi-CG (typically 10^{-4} to 10^{-6})

Output Variables:

ci (complex) - Solution vector of length *nunkns*

icount (integer) - Number of iterations needed for bi-CG method to converge

rerr (real) – Residual error. The bi-CG algorithm will stop when the residual error (*rerr*) becomes less than the pre-specified convergence factor (*conv*).

FUNCTION arow(i, vec)

This function provides the *ith* row of matrix **A** and returns a vector, *vec*, of length *nunkns.*

Input: *i (*integer) – *ith* row number of matrix **A**
Output: *vec* (complex vector) – *ith* row of matrix **A** vector of length, *nunkns*.

FUNCTION acol (j, vec)

This function provides the *jth* column of matrix, **A,** and returns a vector, *vec*, of length *nunkns*.

Input: *j* (integer) – *jth* row number of matrix **A**
Output: *vec* (complex vector) – *jth* column of matrix **A** vector of length, nunkns.

COMPUTATION OF HANKEL FUNCTIONS H0 AND H1

The tensor Green's function components require the evaluation of Hankel functions H_0 and H_1. Because the argument of the Hankel functions depends on the relative distance between the source *(x',y')* and observation *(x,y)* point, it was determined that there are only a limited number of observation and source point combinations that are repeated over and over again in the computation of the Green's function. The Hankel functions are computed for different combinations of source and observation points and stored in two arrays. Then the stored values are utilized in the computation of the tensor Green's function, thereby reducing computation time by avoiding repeated computations.

APPLICATION OF bi-CG METHOD TO SOLVE AN INTERGAL EQUATION

Here we provide an example of solving a system of linear equations using the bi-CG method. The system of equations is obtained as part of the numerical solution of an integral equation. The first attempt at solving the equations was done by using the traditional approach of matrix inversion. However this required storing the matrix **A**. As the dimensionality of the problem increased, thereby increasing the number of unknowns as well as the size of matrix **A,** it was clear that the resulting problem could not be solved because of the limited storage capacity of the personal computer: hence the move to bi-CG method.

Consider a scattering system described by a dielectric function, $\varepsilon(r)$, embedded in an infinite homogeneous background material, ε_B. When the system is illuminated by an incident field, $E^0(r)$, the total electric field, $E(r)$, is given by the following integral equation (Reference 4):

$$E(r) = E^0(r) + \int_V dr' G^B(r,r')\ k_0^2\ \Delta\varepsilon(r')E(r')$$

where $G^B(r,r')$ is the Green's tensor associated with the infinite background ε_B, $\Delta\varepsilon(r) = \varepsilon(r) - \varepsilon_B$ and the integration is over the entire scatterer volume, *V*. To implement a numerical solution to the above integral equation, we define a grid with *N* meshes. Each mesh, *i*, is centered at position, r_i, and has a volume, $V_i, i = 1,2,...N$ (for a 2-dimensional systems, V_i will be replaced by the area of the mesh, *i*). Representing the discretized electric field, $E_i = E(r_i)$, the dielectric constant, $\Delta\varepsilon_i = \Delta\varepsilon(r_i)$, and the Green's function, $G^B(i,j) = G^B(r_i,r_j)$, the integral equations can be written as a system of linear equations:

The Green's tensor $G^B(r,r')$ is given by

$$G^B(r,r') = \begin{bmatrix} [G^B_{xx}] & [G^B_{xy}] & [G^B_{xz}] \\ [G^B_{xy}] & [G^B_{yy}] & [G^B_{yz}] \\ [G^B_{xz}] & [G^B_{yz}] & [G^B_{zz}] \end{bmatrix}$$

For a complete description of the Green's tensor components, refer to Reference 4. For a geometry with *N* meshes, *N= nx*ny,* where *nx* and *ny* represent the number of divisions of the scatterer in *x* and *y* directions, respectively, the total number of unknowns is 3**N* to account for the three components of the electric field. Each of the sub-matrices in the tensor Green's function is now of the order of *NxN*. Now we provide a 2-dimensional scattering problem that uses the above formulation to compute the total electric field.

NUMERICAL EXAMPLE

We provide a numerical example (Reference 5) utilizing the formulation presented in this Section. We consider a metal wire of radius = *10 nm* with a dielectric constant, $\varepsilon = (-1.07, 0.29)$. The wavelength of the incident field is $\lambda = 388\,nm$. The geometry is discretized in a grid of 250*x*250 (*nx=ny=250*), resulting in a total of *125,000* (2**nx***ny*) unknowns. The number of unknowns in this example is 2**nx***ny,* as only two components of the electric field are needed due to the polarization of the incident field. (Note that if we needed to store the matrix **A** of dimension (125,000x125,000), the storage space needed would be very large. This provides the motivation for using the bi-

CG iterative method, in which case we do not store this matrix). Figure 1 shows the electric field amplitude relative to the incident field around the wire as it is illuminated with a plane wave.

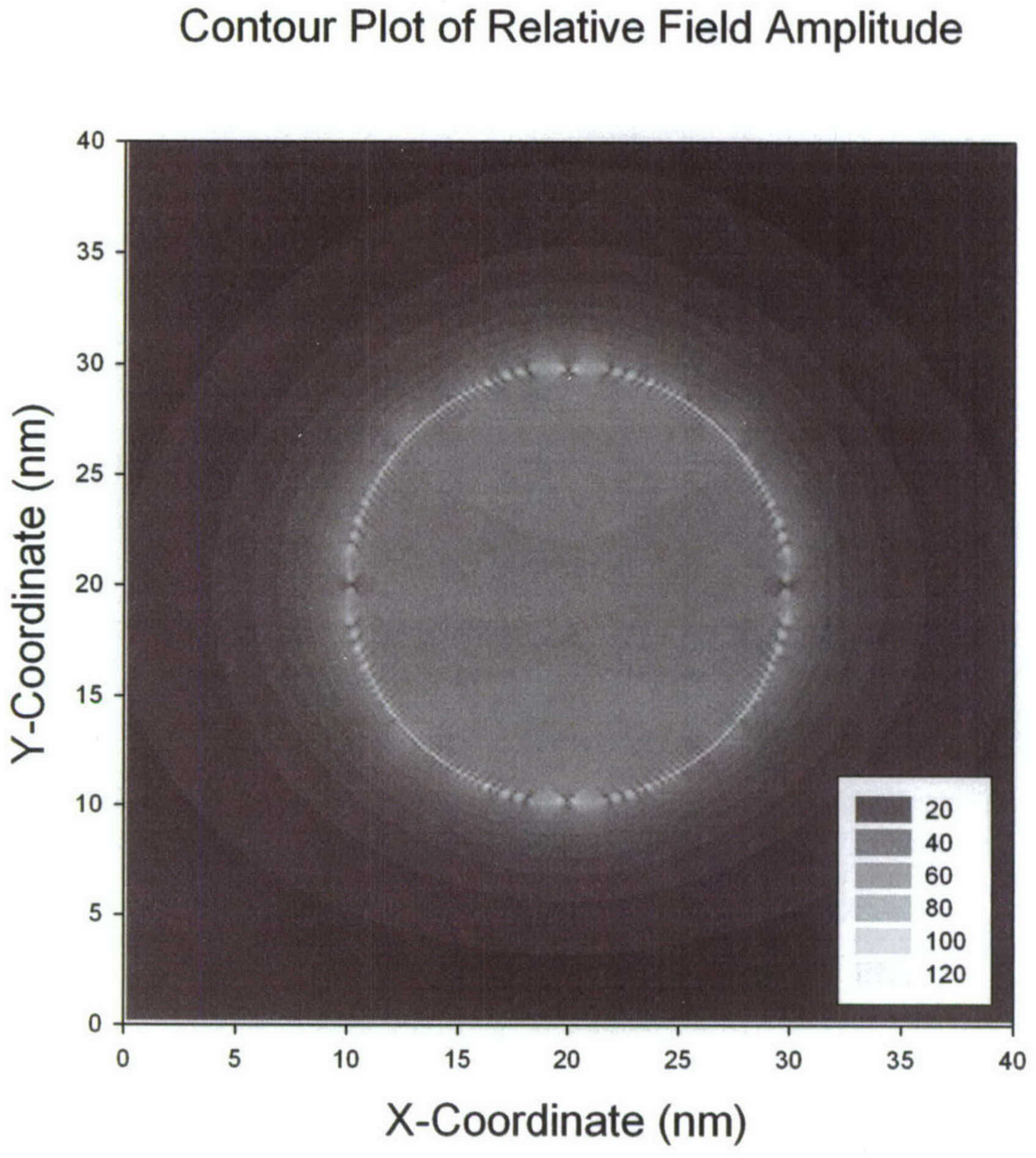

FIGURE 1. Contour Plot of Relative Field Amplitude.

REFERENCES

1. T. K. Sarkar and S. M. Rao. "The Application of Conjugate Gradient Method for the Solution of Electromagnetic Scattering From Arbitrarily Oriented Wire Antennas," *IEEE Trans. Antennas Propag.*, Vol. 32, No. 4 (April 1994), pp. 398-403.

2. H. A. Van Der Vorst. "Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Non-symmetric Linear Systems," *SIAM J. Sci. and Statistical Computing*, Vol. 13, No. 2 (March 1992), pp. 631-44.

3. Carnegie Mellon University. *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*, by J. R. Shewchuck. Pittsburgh, Pennsylvania, CMU School of Computer Science (http://www.cs.berkley.edu/~jrs/).

4. O. J. F. Martin and N. B. Piller. "Electromagnetic Scattering in Polarizable Backgrounds," *Phys. Rev. E*, Vol. 58, No. 3 (September 1998), pp. 3909-15.

5. J. P. Kottmann and O. J. F. Martin. "Plasmon Resonances of Silver Nanowires With a Nonregular Cross Section," *Phys. Rev. B*, Vol. 64 (8 November 2001) 235402-1:235402-10.

Appendix
FORTRAN CODE OF THE BI-CG ALGORITHM AND ASSOCIATED FUNCTION SUBPROGRAMS

Here is the listing of the computer code with subroutine *bcg* and function subprograms *arow* and *acol:*

```
   !     ------------------------------------------------------------------
!
!      bi-conjugant gradient (works)
!      solution to integral equation for e-field
!
!
!      ----------------------------------------------------------------
      program bcg_method
      implicit none
      real,parameter :: system_size=40 !total grid size in nm
      real, parameter :: lambda0=338 !incident wavelength in nm
      real, parameter :: L=system_size/lambda0 !total system size in
wavelengths
      integer, parameter :: nx=60, ny=nx, nxy = nx*ny !total number of grid
points
      integer, parameter :: maxi=3*nxy
      complex, parameter :: Im=(0., 1.),zero=(0., 0.),one=(1., 0.)
      complex, dimension(1:3*nxy) :: E0,etotal
      integer, dimension(1:3*nxy) :: ipiv
      complex, dimension(1:2*nxy) :: H0,H1
      integer, dimension(1:nxy) :: ix,jy
      complex, dimension(1:nxy) :: eps
      real:: arg, r_eff,theta,psi,krho,phi,pi,kx,ky,x,y,wc,wcda,dx,dy
      real :: xd,yd,xp,yp
      complex :: mfactor,Mx,My,Mz,kz,eb,z1,mxx,myy,mzz
      integer :: i,j,k,icount,ij
      real::conv,rerr,rho1,rho2,rho3,rho
      INTEGER :: clock_start,clock_end,clock_rate
      REAL  :: elapsed_time,f
      real :: r1
   CALL SYSTEM_CLOCK(COUNT_RATE=clock_rate) ! Find the rate
   CALL SYSTEM_CLOCK(COUNT=clock_start) ! Start timing
      conv=.0001

      E0=0.
      eb=one
      dx=L/nx
      dy=dx
      !dy=.025    !grid size per wavelength
      pi = acos(-1.)
      wc = 2.*pi
      wcda = wc**2*dx*dy
      theta=90.
     !  phi = 90.
     ! psi=0.
      kx=-cos(theta*pi/180.)
      ky=-sin(theta*pi/180.)
      kz=0.
      krho=sqrt(eb) !krho/k0
      r_eff = sqrt(dx*dy/pi)
      mfactor=wc*r_eff*H1_fun(wc*r_eff*krho)/krho+2*Im/(pi*krho**2)
      Mx = Im*pi/4*(2.-krho**2)*mfactor
      My=Mx
      Mz = Im*pi/2*(1.-kz**2)*mfactor

 k = 0
      do i = 1, nx
       do j = 1, ny
       k = k + 1 !k goes up to ny*nx
       x=i*dx
       y=j*dy
       ix(k) = i
       jy(k) = j
       arg=kx*x+ky*y
       z1=exp(wc*Im*arg)
    !   E0(k) =
z1*(cos(phi*pi/180.)*sin(theta*pi/180.)*cos(psi*pi/180.)-&
     !           sin(phi*pi/180.)*sin(psi*pi/180.))
      ! E0(k+nxy) =
z1*(sin(phi*pi/180.)*sin(theta*pi/180.)*cos(psi*pi/180.)+&
       !           cos(phi*pi/180.)*sin(psi*pi/180.))
       !E0(k+2*nxy) = z1*cos(theta*pi/180.)* cos(psi*pi/180.)
      E0(k+2*nxy) = z1
      end do
       end do


 eps = zero
 r1=10./lambda0 !radius in wavelengths
!  choose sites of delta_epsilon over 2D grid
    k = 0
    do i = 1,nx
     do j = 1,ny
      k = k + 1
     eps(k)=zero
      ! eps(k) = one*1.5
       rho1=sqrt(real(i-nx/2)**2+real(j-ny/2)**2)
       if (rho1<=int(r1/dx)) eps(k) = (-1.07,0.29) - one
!        rho2=sqrt(real(i-nx/2)**2+real(j-ny/2)**2)
!        if (rho2<=3.) eps(k) = (-100.,0.) - one
!        rho3=sqrt(real(i-3*(nx/4))**2+real(j-ny/2)**2)
!        if (rho3<=3.) eps(k) = (-100.,0.) - one
!      if(i >= nx/2-5 .and. i <= nx/2+5 .and. &
!          j >= ny/2-5 .and. j <= ny/2+5) eps(k) = (-100.,0.) - one
     ! write(45,445) ix(k),jy(k),real(eps(k))
      end do
       end do
      !close(45)


   i=1
do j=2,nxy
```

```
x  = ix(i)*dx;y=jy(i)*dy
      xp = ix(j)*dx;yp = jy(j)*dy
      xd=x-xp;yd=y-yp
      rho = sqrt(xd*xd+yd*yd)
      arg = wc*krho*rho
      ij=(ix(i)-ix(j))**2 + (jy(i)-jy(j))**2
 H0(ij)=H0_fun(arg)
 H1(ij)=H1_fun(arg)
 end do

mxx=Mx-0.5
myy=My-0.5
mzz=Mz
!***********************************************************************
******

       call bcg(E0,3*nxy,maxi,conv,etotal,icount,rerr)
      print *,'number of iterations=',icount-1
      print *,'the residual error =',rerr
     ! print *,testx
!***********************************************************************
******

      do k = 1, nxy !Ex
      x = ix(k)*dx ; y = jy(k)*dy
      write(97,*) x*lambda0, y*lambda0, abs(etotal(k))
      end do

      do k = nxy+1, 2*nxy !Ey
      x = ix(k-nxy)*dx ; y = jy(k-nxy)*dy
      write(98,*) x*lambda0, y*lambda0, abs(etotal(k))
      end do

      do k = 2*nxy+1, 3*nxy !Ez
      x = ix(k-2*nxy)*dx ; y = jy(k-2*nxy)*dy
      write(99,*) x*lambda0, y*lambda0, abs(etotal(k))
      end do


     do k=1,nxy
      x = ix(k)*dx ; y = jy(k)*dy

arg=etotal(k)*conjg(etotal(k))+etotal(k+nxy)*conjg(etotal(k+nxy))+etotal
(k+2*nxy)*conjg(etotal(k+2*nxy))
      write(96,*) x*lambda0, y*lambda0, arg
      end do
!***********************************************************************
******

444  format(F6.3,1X,F6.3,1X,F6.3)


CALL SYSTEM_CLOCK(COUNT=clock_end) ! Stop timing
   ! Calculate the elapsed time in seconds:
   elapsed_time=REAL((clock_end-clock_start))/clock_rate
   print *,'elapsed time =',elapsed_time/60.,'minutes'
      contains

subroutine bcg(b,nunkns,nitm,conv,ci,nit,rerr)
implicit none
complex, intent(in), dimension(1:nunkns) :: b
```

```
complex, intent(out), dimension(1:nunkns) :: ci
integer, intent(in) :: nunkns,nitm
integer, intent(out) :: nit
real, intent(in) :: conv
real, intent(out) :: rerr
complex,dimension(1:nunkns) :: x,p,pn,r,rn
complex :: beta,alpha,val1,val2,val3,val4,val5,sumx
integer ::i

x=0;p=0;pn=0;ci=0;icount=0;beta=0
rerr=100
r=b
rn=conjg(r)
val1=dot_product(conjg(b),b)
 do nit=1,nitm

   if (rerr>conv) then
    val2=dot_product(conjg(r),r)
    rerr=abs(val2/val1)
    val3=dot_product(conjg(rn),r)
    if (nit/=1) beta=val3/val5
    p=r+beta*p
    pn=rn+beta*pn

    forall (i=1:nunkns) x(i)=dot_product(conjg(arow(i)),p)

    val4=dot_product(conjg(pn),x)
    alpha=val3/val4
    ci=ci+alpha*p
    r=r-alpha*x
    forall (i=1:nunkns) x(i)=dot_product(conjg(acol(i)),pn)

    rn=rn-alpha*x
    val5=val3
    icount=icount+1
   print *,'iteration#',nit-1
   print *,'residual error',rerr
   else
   exit
   end if

 end do
end subroutine bcg

function arow(i)
implicit none
integer, intent(in) :: i
complex, dimension(1:3*nxy) :: arow
integer :: j,n

n=nxy

do j=1,3*n

if (i<=n) then

   if (j<=n) then
      if (i==j) then
       arow(j)=1.-mxx*eps(j)
      else
        if (eps(j)==zero) then
```

```
          arow(j)=0.
          else
        arow(j)=-gxx(i,j)*eps(j)
          end if
        end if
     else if (j>n .and. j<=2*n) then
         if (eps(j-n)==zero) then
           arow(j)=0.
           else
           arow(j)=-gxy(i,j-n)*eps(j-n)
           end if
     else if (j>2*n) then
         if (eps(j-2*n)==zero) then
           arow(j)=0.
           else
           arow(j)=-gxz(i,j-2*n)*eps(j-2*n)
           end if
    end if

   end if

   if (i>n .and. i<=2*n) then

     if (j<=n) then
         if (eps(j)==zero) then
           arow(j)=0.
           else
           arow(j)=-gxy(i-n,j)*eps(j)
         end if
     else if (j>n .and. j<= 2*n) then
      if (i==j) then
         arow(j)=1.-myy*eps(j-n)
      else
         if (eps(j-n)==zero) then
           arow(j)=0.
           else
           arow(j)=-gyy(i-n,j-n)*eps(j-n)
           end if
      end if
     else if (j>2*n) then
         if (eps(j-2*n)==zero) then
           arow(j)=0.
           else
         arow(j)=-gyz(i-n,j-2*n)*eps(j-2*n)
         end if
     end if
   end if


   if (i>2*n) then

     if (j<=n) then
         if (eps(j)==zero) then
           arow(j)=0.
           else
         arow(j)=-gxz(i-2*n,j)*eps(j)
         end if

     else if (j>n .and. j<= 2*n) then
         if (eps(j-n)==zero) then
           arow(j)=0.
          else
        arow(j)=-gyz(i-2*n,j-n)*eps(j-n)
        end if
    else if (j>2*n) then
    if (i==j) then
          arow(j)=1.-mzz*eps(j-2*n)
    else
        if (eps(j-2*n)==zero) then
          arow(j)=0.
          else
        arow(j)=-gzz(i-2*n,j-2*n)*eps(j-2*n)
        end if
     end if
    end if
end if


end do
end function arow

function acol(j)
implicit none
integer, intent(in) :: j
complex, dimension(1:3*nxy) :: acol
integer :: n,i

n=nxy
do i=1,3*n


if (i<=n) then

    if (j<=n) then
       if (i==j) then
        acol(i)=1.-mxx*eps(j)
       else
        if (eps(j)==zero) then
          acol(i)=0.
          else
        acol(i)=-gxx(i,j)*eps(j)
        end if
       end if
    else if (j>n .and. j<=2*n) then
        if (eps(j-n)==zero) then
          acol(i)=0.
          else
        acol(i)=-gxy(i,j-n)*eps(j-n)
        end if
    else if (j>2*n) then
        if (eps(j-2*n)==zero) then
          acol(i)=0.
          else
        acol(i)=-gxz(i,j-2*n)*eps(j-2*n)
        end if
  end if
end if

if (i>n .and. i<=2*n) then

    if (j<=n) then
        if (eps(j)==zero) then
```

```
            acol(i)=0.
            else
          acol(i)=-gxy(i-n,j)*eps(j)
          end if

      else if (j>n .and. j<= 2*n) then
       if (i==j) then
          acol(i)=1.-myy*eps(j-n)
       else
          if (eps(j-n)==zero) then
            acol(i)=0.
            else
          acol(i)=-gyy(i-n,j-n)*eps(j-n)
          end if
       end if
      else if (j>2*n) then
          if (eps(j-2*n)==zero) then
            acol(i)=0.
            else
          acol(i)=-gyz(i-n,j-2*n)*eps(j-2*n)
          end if
      end if
    end if


    if (i>2*n) then

      if (j<=n) then
          if (eps(j)==zero) then
            acol(i)=0.
            else
          acol(i)=-gxz(i-2*n,j)*eps(j)
          end if

      else if (j>n .and. j<= 2*n) then
          if (eps(j-n)==zero) then
            acol(i)=0.
            else
          acol(i)=-gyz(i-2*n,j-n)*eps(j-n)
          end if
      else if (j>2*n) then
      if (i==j) then
          acol(i)=1.-mzz*eps(j-2*n)
        else
          if (eps(j-2*n)==zero) then
            acol(i)=0.
            else
          acol(i)=-gzz(i-2*n,j-2*n)*eps(j-2*n)
          end if
       end if
      end if
    end if


    end do
    end function acol


    complex function gxx(i,j)
          implicit none
      integer,intent(in) :: i,j
      integer :: ij,dif1,dif2
      real :: rho,x,y,xp,yp,rx,ry,c2theta,xd,yd

      x  = ix(i)*dx;y  = jy(i)*dy
      xp = ix(j)*dx;yp = jy(j)*dy
      xd=x-xp;yd=y-yp
      rho = sqrt(xd*xd+yd*yd)
      dif1= ix(i)-ix(j);dif2=jy(i)-jy(j)
      ij=dif1*dif1+dif2*dif2
      rx = xd/rho
      ry = yd/rho
      c2theta=rx*rx-ry*ry
      gxx=wcda*Im/4.*((1.-
krho*rx*krho*rx)*H0(ij)+krho*c2theta*H1(ij)/(wc*rho))

end function gxx

complex function gxy(i,j)
      implicit none
      integer,intent(in) :: i,j
      integer :: ij,dif1,dif2
      real :: rho,x,y,xp,yp,rx,ry,s2theta,xd,yd
      if(i /= j) then
      x  = ix(i)*dx;y=jy(i)*dy
      xp = ix(j)*dx;yp=jy(j)*dy
      xd=x-xp;yd=y-yp
      rho = sqrt(xd*xd+yd*yd)
      arg = wc*krho*rho
      dif1= ix(i)-ix(j);dif2=jy(i)-jy(j)
      ij=dif1*dif1+dif2*dif2
      rx = xd/rho
      ry = yd/rho
      s2theta=2*ry*rx
     gxy=wcda*Im/8.*krho*krho*s2theta*(2./arg*H1(ij)-H0(ij))
      else
      gxy = zero
      end if
end function gxy

complex function gxz(i,j)
      implicit none

      integer,intent(in) :: i,j
      integer :: ij,dif1,dif2
      real :: rho,x,y,xp,yp,rx,xd,yd
      if(i /= j) then
      x=ix(i)*dx ; y=jy(i)*dy
      xp=ix(j)*dx ; yp=jy(j)*dy
      xd=x-xp;yd=y-yp
      rho = sqrt(xd*xd+yd*yd)
      arg = wc*krho*rho
      dif1= ix(i)-ix(j);dif2=jy(i)-jy(j)
      ij=dif1*dif1+dif2*dif2
      rx = xd/rho
      gxz=wcda*1./4.*krho*kz*rx*H1(ij)
      else
      gxz = zero
      end if
end function gxz
```

```
complex function gyy(i,j)
      implicit none
      integer,intent(in) :: i,j
      integer :: ij,dif1,dif2
      real :: rho,x,y,xp,yp,rx,ry,c2theta,xd,yd
      x =ix(i)*dx ;y=jy(i)*dy
      xp=ix(j)*dx;yp=jy(j)*dy
      xd=x-xp;yd=y-yp
      rho = sqrt(xd*xd+yd*yd)
      arg = wc*krho*rho
      dif1= ix(i)-ix(j);dif2=jy(i)-jy(j)
      ij=dif1*dif1+dif2*dif2
      rx = xd/rho
      ry = yd/rho
      c2theta=rx*rx-ry*ry
      gyy=wcda*Im/4.*((1.-krho*ry*krho*ry)*H0(ij)-
krho*c2theta*H1(ij)/(wc*rho))

end function gyy

complex function gyz(i,j)
      implicit none
      integer,intent(in) :: i,j
      integer :: ij,dif1,dif2
      real :: rho,x,y,xp,yp,ry,xd,yd
      if(i /= j) then
      x=ix(i)*dx ;y = jy(i)*dy
      xp=ix(j)*dx ;yp= jy(j)*dy
      xd=x-xp;yd=y-yp
      rho = sqrt(xd*xd+yd*yd)
      arg=wc*krho*rho
      dif1= ix(i)-ix(j);dif2=jy(i)-jy(j)
      ij=dif1*dif1+dif2*dif2
      ry=yd/rho
      gyz=wcda*1./4.*krho*kz*ry*H1(ij)
      else
      gyz = zero
      end if
end function gyz

complex function gzz(i,j)
      implicit none
      integer,intent(in) :: i,j
      integer :: ij,dif1,dif2
      real :: rho,x,y,xp,yp,xd,yd
      x  = ix(i)*dx ; y  = jy(i)*dy
      xp = ix(j)*dx ; yp = jy(j)*dy
      xd=x-xp;yd=y-yp
      rho = sqrt(xd*xd+yd*yd)
      arg = wc*krho*rho
      dif1= ix(i)-ix(j);dif2=jy(i)-jy(j)
      ij=dif1*dif1+dif2*dif2
      gzz= wcda*Im/4.*(1.-kz*kz)*H0(ij)

end function gzz


FUNCTION H0_fun(z) RESULT (H0out)
IMPLICIT NONE
real, INTENT(IN) :: z
real:: x,J_0,Y_0,f_0,theta,x2,x3,x4,x5,x6,x8,x10,x12,sqz
```

```
complex::H0out

if (z <= 3.) then
x=z/3.
x2=x*x;x4=x2*x2;x6=x4*x2;x8=x6*x2;x10=x8*x2;x12=x10*x2
J_0=1.-2.2499997*x2+1.2656208*x4-.3163866*x6+.0444479*x8-
.0039444*x10+.0002100*x12
Y_0=(2/pi)*log(z/2.)*J_0+.36746691+.60559366*x2-
.74350384*x4+.25300117*x6-.04261214*x8+.00427916*x10-.00024846*x12
else
x=3./z
x2=x*x;x3=x2*x;x4=x3*x;x5=x4*x;x6=x5*x;sqz=sqrt(z)
f_0=.79788456-.00000077*x-.00552740*x2-.00009512*x3+.00137237*x4 &
   -.00072805*x5 + .00014476*x6

theta=z-.78539816-.04166397*x-.00003954*x2+.00262573*x3-.00054125*x4 &
&   -.00029333*x5+.00013558*x6

J_0=f_0*cos(theta)/sqz
Y_0=f_0*sin(theta)/sqz
end if

H0out=cmplx(J_0,Y_0)
END FUNCTION H0_fun


FUNCTION H1_fun(z) RESULT (H1out)
IMPLICIT NONE
real, INTENT(IN) :: z
real:: x,J_1,Y_1,f_1,theta1,x2,x3,x4,x5,x6,x8,x10,x12,sqz
complex::H1out

if (z <= 3.) then
x=z/3.
x2=x*x;x4=x2*x2;x6=x4*x2;x8=x6*x2;x10=x8*x2;x12=x10*x2
J_1=.5*z-.56249985*z*x2+.21093573*z*x4-.03954289*z*x6+.00443319*z*x8-
.00031761*z*x10+.00001109*z*x12

Y_1=(2/pi)*log(z/2.)*J_1-.6366198/z+.2212091*x2/z+2.1682709*x4/z-
1.3164827*x6/z+.3123951*x8/z-.0400976*x10/z+.0027873*x12/z
else
x=3./z
x2=x*x;x3=x2*x;x4=x3*x;x5=x4*x;x6=x5*x;sqz=sqrt(z)
f_1=.79788456+.00000156*x+.01659667*x2+.00017105*x3-
.00249511*x4+.00113653*x5-.00020033*x6

theta1=z-2.35619449+.12499612*x+.00005650*x2-
.00637879*x3+.00074348*x4+.00079824*x5-.00029166*x6

J_1=f_1*cos(theta1)/sqz
Y_1=f_1*sin(theta1)/sqz
end if

H1out=cmplx(J_1,Y_1)
END FUNCTION H1_fun


end program bcg_method
```

INITIAL DISTRIBUTION

1	Naval War College, Newport (1E22, President)
1	Headquarters, 497 INOT, Falls Church (IOG Representative)
2	Defense Technical Information Center, Fort Belvoir
1	Center for Naval Analyses, Alexandria, VA (Technical Library)
8	Surenda Singh, Department of Electrical Engineering, The University of Tulsa, 600 S. College Avenue, Tulsa OK 74104

ON-SITE DISTRIBUTION

4	Code 4BL000D (3 plus Archives copy)
1	Code 4T0000D, Markarian
12	Code 4T4110D
	Chesnut (1)
	Elson (5)
	Halterman (5)
	Overfelt (1)